Keyframe-Based Video Object Deformation

Yanlin Weng, Weiwei Xu, Shichao Hu, Jun Zhang, and Baining Guo

Abstract—This paper proposes a keyframe-based video object editing scheme for automatic object shape deformation. Except for object segmentation, several technologies are developed in the proposed scheme to minimize user interactions as well as to facilitate users' flexible and precise control on deformation. First, an automatic modeling technique is presented to establish the graph model for the segmented object, which can accurately represent 2D shape information. Second, when the user specifies the deformation of a video object in keyframes by dragging handles on it to new positions, an algorithm is proposed to automatically generate motion trajectories of the object handles along with frames. Finally, the 2D shape deformation is completed by our proposed non-linear energy minimization algorithm. Furthermore, in order to handle the abrupt change of positions, dimensionality reduction is applied together with the minimization algorithm. Experimental results fully demonstrate that the proposed scheme can generate natural video shape deformations with few user interactions.

Index Terms—Video Object, Shape Deformation, Shape Editing.

1 Introduction

Video editing is the process of re-arranging or modifying segments of video to form another piece of video. It has long been used in the film industry to produce studio-quality motion pictures. With the recent advances in interactive video segmentation, it is much easier to cut out dynamic foreground objects from a video sequence [4], [8], [11], [17]. Nowadays, video editing has been focused at the object level. An example operation is video object cut and paste, which is widely used in movie production to seamlessly integrate a video object into new scenes for special effects.

Recently, many research efforts have been devoted to video object editing. Candemir et al. [15] present a texture replacement technique for video objects using a 2D mesh-based mosaic representation. Their technique can handle video objects with self or object-to-object occlusion. Non-photorealistic rendering (NPR) techniques have been generalized to render video objects in cartoon style [1], [19]. For the editing of motion, Liu et al. [12] present a motion magnification technique to amplify the small movements of video objects. Wang et al. [18] use a cartoon animation filter to exaggerate the animation of video objects to create stretch and squash effects. All these techniques are very useful to generate new video objects with different textures, styles or motions, and achieve interesting video effects.

This paper focuses on video object shape deformation. There already exist various previous works on shape editing of static objects in an image. Barrett et al. [3] propose an object-based image editing system, which allows the user to animate a static object in an image. Recent 2D shape deformation algorithms aim to produce visually pleasing results with shape feature preservation and to provide interactive feedback to users. Igrashi et al. [7] develop an interactive system that allows the user to deform a 2D triangular mesh by manipulating a few points. To make the deformation as-rigid-as-possible [2], they present a two-step linearization algorithm to minimize the distortion of each triangle. However, it might cause unnatural deformation results due to its linear nature. On the other hand, the algorithm based on moving least squares [14] does not require a triangular mesh and can be applied to general images. The 2D shape deformation algorithm in [20] solves the deformation using nonlinear least-squares optimization. It tries to preserve two geometric properties of 2D shapes: the Laplacian coordinates of the boundary curve of the shape and local areas inside the shape. The resulting system is able to achieve physically plausible deformation results and runs in real time. Our work is also inspired by the recent research trend on generalizing static mesh editing techniques to the editing of mesh animation data [9], [21]. We wish to generalize static image object editing techniques to deform video objects.

In this paper, we present a novel video object deformation scheme for shape editing of video objects. Our scheme has the following features:

- **Keyframe-based editing:** Our scheme has a keyframe-based user interface. The user only needs to manipulate the video object at several keyframes. At each keyframe, the user deforms the 2D shape in the same way as in traditional image deformation. Our algorithm will smoothly propagate the deformation result from the keyframes to the remaining frames and automatically generate the new video object. In this way, it is able to minimize the amount of user interaction, while providing flexible and precise user control.
- **Temporal coherence preservation:** Our algorithm can preserve the temporal coherence of the video object in the original video clip.
- **Shape feature preservation:** Our algorithm is effective in preserving shape features of the video object while generating visually pleasing deformation.

To develop such a scheme, we need to address the following

E-mail:weng@uwm.edu, University of Wisconsin - Milwaukee
E-mail:wwxu@microsoft.com, Microsoft Research Asia
E-mail:shichaoh@microsoft.com, Microsoft Corporation
E-mail:junzhang@uwm.edu, University of Wisconsin - Milwaukee
E-mail:bainguo@microsoft.com, Microsoft Research Asia

challenges.

First, we need a shape representation for video objects. Unlike a static image object, a video object is dynamic in nature. Therefore, its shape is changing with time. We first use recent interactive video cutout tools [11], [17] to extract the foreground video object. Then a keyframe-based contour tracking technique [1] is employed to locate the boundary Bézier curves of the video object. The shape of the video object is subsequently represented as the region bounded by these boundary curves.

Secondly, to preserve the temporal coherence of the video object, the keyframe editing results need to be smoothly propagated from the keyframes to the remaining frames such that the whole video object can be deformed consistently. In our system, the user manipulates the deformation handles to deform the shape at each keyframe, and we use a least-squares handle editing propagation algorithm to create a new motion trajectory of the handles along with frames according to the handle editing results at keyframes. Therefore, after handle editing propagation, we can get the target positions of the handles at each frame. Then 2D shape deformation can be applied to each frame given the propagated handle constraints. This handle editing propagation algorithm can elegantly preserve temporal coherence because it tries to minimize an energy function that represents the temporal properties of the original motion trajectory of the handles.

Finally, although the nonlinear 2D shape deformation algorithm described in [20] can preserve shape features and achieve physically plausible results in interactive keyframe editing, it may generate unsatisfactory results for the remaining frames with the propagated handle constraints (see Fig. 4). This is because the deformation solver may stick to a bad local minimum due to the abrupt change of handle positions in the remaining frames, while the handle positions are smoothly changed in keyframe editing. We therefore propose a dimensionality reduction technique to project the deformation energy onto the subspace formed by the control points of the boundary Bézier curves. Performing energy minimization in this subspace greatly improves the stability and convergence of the nonlinear deformation process, and thus leads to much better deformation results.

The remainder of this paper is organized as follows. Algorithm overview and the details of each step are presented in Section 2. In Section 3, we introduce how we handle video objects with self-occlusion. Experimental results are shown in Section 4, and the paper concludes with some discussion of future work in Section 5.

2 Video Object Deformation

2.1 Algorithm Overview

As illustrated in Figure 1, our video object deformation algorithm performs the following steps in a typical editing session: video object cutout, video object shape generation, keyframe editing, deformation propagation and new video object generation.

Given an input video, the video object cutout step extracts the foreground object from the video using the segmentation

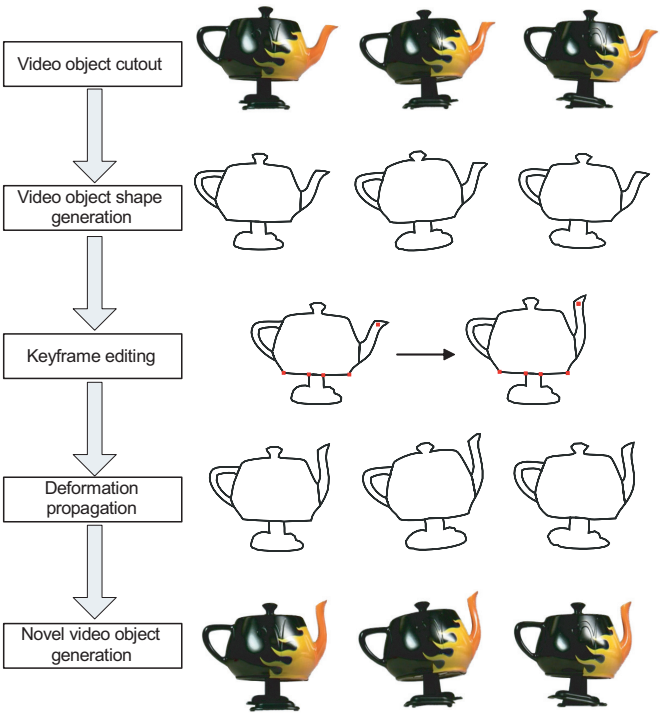

Fig. 1. Main steps of our video object deformation system.

based video cutout technique in [11]. The result is a sequence of foreground images with alpha channels indicating the opacity of each pixel.

However, the foreground images cannot be directly used by the shape deformation algorithm. In the second step, video object shape generation, we aim to represent the foreground images with a sequence of 2D shapes and corresponding textures, which can be the input of the deformation algorithm. To build a consistent shape presentation along the time dimension, the keyframe based rotoscoping technique in [1] is used to track the contours of the video object in the foreground images. We first manually draw several Bezier curves along the boundary of the video object at some keyframes. Then the tracking algorithm will locate the optimal position of the curves in the remaining frames. These Bezier curves are then organized into a boundary polygon, and our algorithm automatically inserts a set of points into the interior region of the polygon and generates a 2D graph by connecting the vertices of the boundary polygon and the inside points (see Section 2.2 for details). Finally, we get a sequence of 2D shapes: $\{S^i, i = 1,...,N\}$, and the non-transparent part of foreground images are stored as textures for each frame.

With the generated sequence of 2D shapes: $\{S^i, i = 1,...,N\}$, keyframe editing is the next step in which users edit the keyframes. The editing results are used to instruct the video object deformation algorithm to automatically create novel video objects. During keyframe editing, the user can deform the 2D shape S^k at any frame k. The edited frame will subsequently become a keyframe. During deformation, the user only needs to specify deformation handles $\{H_j^k, j = 1,...,l\}$ at frame k, and then drag the handles to new positions $\hat{H}_j^k$. The 2D shape

deformation algorithm in [20] can be used to automatically deform the shape S^k into $\hat{S}^k$.

Deformation propagation is the the most important step of our algorithm, since it will automatically generate a novel 2D shape sequence according to the keyframe editing result. It first propagates the user edited handle $\hat{H}_j^k$ at keyframe k to the remaining frames to compute the new position of handle $\hat{H}_j^i$ at each frame i, and then applies the shape deformation algorithm at each frame to meet the new positional constraints from handles $\hat{H}_j^i$ and simultaneously generates the deformed 2D shape $\hat{S}^i$. To preserve the temporal properties of the motion trajectories of the handles in the original video sequence, a least-squares optimization algorithm is proposed for handle editing propagation, and a novel 2D shape deformation algorithm based on dimensionality reduction is presented to guarantee the quality of the shape deformation result.

The final step, new video object generation, uses the foreground images as textures to render the new 2D shapes $\{\hat{S}^i, i = 1, ..., N\}$ to produce the new foreground images, which composes a novel video object ready for integration into any new background image or video.

Note that the contour tracking in the shape generation step is a challenging task in computer vision because of the complexity of motion. By applying it to foreground images only, we are able to reduce the difficulty greatly. Our system also supports the deformation of video objects with self-occlusions. To achieve this, we model the shape of video objects with multiple polygons which may occlude each other. Please see Section 3 for details.

2.2 Video Object Shape Representation

The output of the contour tracking algorithm is the positions of the Bézier curves at each frame. We need to make use of these curves to construct a 2D shape representation for the video object, which is actually a 2D graph suitable for the deformation algorithm. Instead of asking the user to tediously do this frame by frame, we first build a 2D graph for the video object with some user interaction in the first frame. Then our system will automatically transfer the graph to the remaining frames.

As shown in Figure 2(a), after tracking, the boundary of the walking teapot in the first frame is represented with Bézier curves. In this paper, we use cubic Bézier curves:

$$\mathbf{g}(t) = \mathbf{p}_0(1-t)^3 + 3\mathbf{p}_1 t(1-t)^2 + 3\mathbf{p}_2 t^2(1-t) + \mathbf{p}_3 t^3, \quad (1)$$

where $\mathbf{p}_i$ is the control points, and $t \in [0,1]$ is a scalar parameter.

The Bézier curves are then automatically discretized into connected line segments by uniformly sampling the parameter t (Figure 2(b)).

To define the interior region of the shape, the user needs to specify how the line segments are arranged into polygons. In Figure 2(c), we organize the line segments into four polygons. In this way, we can manipulate the four components of the teapot: the handle, the body, the feet and the spout. As in [20], a few interior points are then inserted into each polygon and then connected to form the 2D graphs (Figure 2(d)).

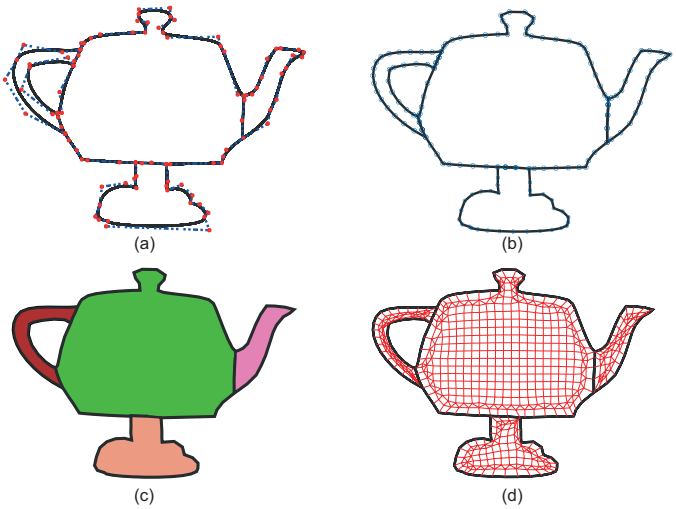

Fig. 2. Video object shape generation. (a) Boundary Bézier curves. The control points are shown as red dots. (b) Discretization of Bézier curves. The blue dots indicate the sampled points. (c) Organized polygons (indicated in different colors). (d) Final 2D Graph.

Formally, we can denote the 2D graph as $\{\mathbf{V}^0, \mathbf{E}^0\}$, where $\mathbf{V}^0$ is the set of n vertices in the graph, and $\mathbf{E}^0$ is the set of edges. $\mathbf{V}^0$ consists of two parts: $\mathbf{V}_p^0$ contains m vertices on the shape boundary, and $\mathbf{V}_g^0$ contains $n-m$ interior vertices of the graph.

For each point $v_{g,i}^0$ in $\mathbf{V}_g^0$, we can compute its mean value coordinates [5] in the polygon formed by $\mathbf{V}_p^0$:

$$w_{i,j} = \frac{\tan(\alpha_j/2) + \tan(\alpha_{j+1}/2)}{|v_{g,i}^0 - v_{p,j}^0|},$$

where α_j is the angle formed by the vector $v_{p,j}^0 - v_{g,i}^0$ and $v_{p,j+1}^0 - v_{g,i}^0$. Normalizing each weight function $w_{i,j}$ by the sum of all weight functions yields the mean value coordinates of $v_{g,i}^0$ with respect to $\mathbf{V}_p^0$. Then $v_{g,i}^0$ can be easily represented as a linear combination of $\mathbf{V}_p^0$ according to:

$$v_{g,i}^0 = \sum_{v_{p,j}^0 \in \mathbf{V}_p^0} w_{i,j} * v_{p,j}^0, \quad (2)$$

which can also be represented in matrix form:

$$\mathbf{V}_g^0 = \mathbf{M}_p \mathbf{V}_p^0, \quad (3)$$

where $\mathbf{M}_p$ is a matrix of mean value coordinates.

The transfer algorithm is relatively simple. For each frame i, since each Bézier curve has its corresponding curve in the first frame, we can easily get the boundary polygons (i.e., $\mathbf{V}_p^i$) through sampling the Bézier curves with the same parameters t as in the first frame. Then we directly copy the interior points and edge information from the 2D graph in the first frame, and calculate the interior vertex positions $\mathbf{V}_g^i$ using the same mean value coordinates $\mathbf{M}_p$ computed in the first frame:

$$\mathbf{V}_g^i = \mathbf{M}_p \mathbf{V}_p^i. \quad (4)$$

As a result, the shape of the video object is represented as a sequence of 2D graphs $\{\mathbf{V}^i, \mathbf{E}^i\}, i = 1, ..., N$ (see the companion video). These graphs differ in positions, but have the same topology. Now the user can deform the shape at any keyframe using the algorithm in [20].

2.3 Handle Editing Propagation

During keyframe editing, the user selects some vertices as deformation handles and drags them to new positions. The updated positions of the handles will serve as positional constraints to drive the 2D shape deformation algorithm. Therefore, a handle editing propagation algorithm is necessary to smoothly propagate the handle editing results from the keyframes to the remaining frames such that the 2D shape at each frame can be deformed properly. In order to preserve the temporal properties of the motion trajectories of the handles in the original sequence of 2D graphs, we formulate the handle propagation as a least squares optimization problem as in [21], and the motion trajectory of a handle is abstracted as the temporal curve of the center of the handle.

Since a handle may contain multiple vertices, we define a single local coordinate frame for each handle to force all vertices inside this handle to move together rigidly, which prevents severe distortion of the overall shape of the handle. Formally, let us denote a user specified handle i at frame k as $H_i^k = \{\mathbf{c}_i^k, \mathbf{F}_i^k, \{\mathbf{v}_{i_j}^k | j = 0, ..., n_i - 1\}\}$, where $\{v_{i_j}^k | j = 0, ..., n_i - 1\}$ is the set of vertices inside this handle, and $\mathbf{c}_i^k$ and columns of $\mathbf{F}_i^k$ define the center and axes of the local coordinate frame. $\mathbf{c}_i^k$ can be the centroid of the vertices, and the initial axes can be defined arbitrarily since we only consider the rotation between the initial and altered axes. Note that once a handle is specified at some keyframes by the user, we can easily get a set of corresponding handles in all frames by using the same indices of the vertices inside the handle. The center of all the corresponding handles $\mathbf{c}_i^k, k = 1, ..., N$ forms a curve in the temporal domain, and local coordinate axes $\mathbf{F}_i^k$ are associated with each center (see Figure 3).

The temporal properties of the handle are defined as the rotation invariant coordinate of each center in its neighboring local coordinate system. Precisely, for each center $\mathbf{c}_i^k$, we can compute its rotation invariant coordinate using the following formula in the original animation:

$$\mathbf{c}_i^k - \mathbf{c}_i^l = \mathbf{F}_i^l \mathbf{d}_i^{k \to l}, l \in N^k, \tag{5}$$

where N^k represents the index set of the immediate neighbors of $\mathbf{c}_i^k$, $|N^k| \leq 2$, and $\mathbf{d}_i^{k \to l}$ represents the so-called rotation invariant coordinate of $\mathbf{c}_i^k$. It is obvious that $\mathbf{d}_i^{k \to l}$ is the local coordinate of $\mathbf{c}_i^k$ in its immediate neighboring local coordinate frames.

During handle propagation, we try to preserve $\mathbf{d}_i^{k \to l}$, since it represents the temporal properties computed from the original video. Therefore, we formulate the following quadratic energy for handle propagation:

$$\sum_k \sum_{l \in N^k} \| \hat{\mathbf{c}}_i^k - \hat{\mathbf{c}}_i^l - \hat{\mathbf{F}}_i^l \mathbf{d}_i^{k \to l} \|^2 \tag{6}$$

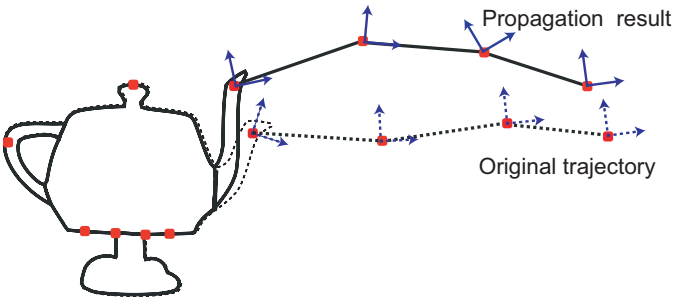

Fig. 3. Handle editing propagation. The dashed line indicates the original temporal curve of the handle, while the solid line indicates the propagation result. A local coordinate frame is associated with each center.

where $\hat{\mathbf{c}}_i^l$ represents the target position of the handle i at frame l, and $\hat{\mathbf{F}}_i^l$ represents the target local coordinate frame axes. Since $\mathbf{d}_i^{k \to l}$ remains the same value computed from Equation (5), the target positions and local coordinate frame axes solved from Equation (6) will have the same temporal properties as those in the input 2D shape animation.

To make Equation (6) a linear least squares problem, we simply interpolate corresponding handle rotations at keyframes over the remaining nodes on the temporal curve to get the orientations of all the new local coordinate frames, $\{\hat{\mathbf{F}}_i^k\}$. Handle rotations are represented as unit quaternions, and the logarithm of the quaternions are interpolated using Hermite splines [10]. The user can optionally designate an influence interval for a keyframe to have finer control over the interpolation.

Once these new local frames are known, the equations in (6) over all unconstrained handle centers give rise to an overdetermined linear system and can be solved using least squares minimization. The new world coordinates of the vertices inside each handle at an intermediate frame can be obtained by maintaining their original local coordinates in the new local coordinate frame at that handle.

2.4 2D Shape Deformation Using Dimensionality Reduction

Once the handle editing is propagated to all frames, we are ready to independently deform the 2D shape at each frame using the nonlinear shape deformation algorithm [20]. However, although the algorithm in [20] works well in interactive keyframe editing, it may produce unsatisfactory results for the remaining frames due to the abrupt change of handle positions in these frames. Inspired by the subspace technique in [6] and model reduction method in [16], we propose a dimensionality reduction technique to project the deformation energy onto the subspace formed by the control points of the boundary Bézier curves. Performing energy minimization in this subspace greatly improves the stability and convergence of the nonlinear deformation process, and thus leads to much better deformation results. In the following, we will first briefly describe the deformation energy, and then introduce how it can be represented as a function of the control points of the Bézier curves only. To facilitate discussion, we omit the frame index in the following equations.

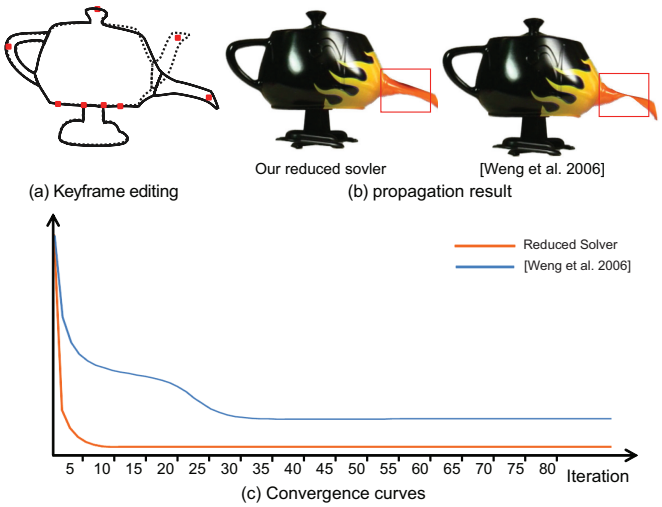

Fig. 4. Comparison between the deformation solver [20] and our dimension-reduced solver. (a) Deforming the shape at a keyframe. The teapot spout is dragged down. The original shape is represented as dashed lines while the deformed shape is represented as solid lines. (b) Propagation result at one frame. The deformation solver in [20] causes serious distortion around the teapot spout, while our dimension-reduced solver generates a more natural result. (c) Convergence curves. The reduced solver converges much faster to an optimal minimum, while the deformation solver in [20] gets stuck in a wrong local minimum.

The deformation energy in [20] can be written as:

$$\|\mathbf{LV}-\delta(\mathbf{V})\|^2+\|\mathbf{MV}\|^2+\|\mathbf{HV}-e(\mathbf{V})\|^2+\|\mathbf{CV}-\mathbf{U}\|^2, \quad (7)$$

where $\mathbf{V}$ is the point positions of the 2D graph, $\|\mathbf{LV}-\delta(\mathbf{V})\|^2$ is the energy term for Laplacian coordinates preservation, $\|\mathbf{MV}\|^2+\|\mathbf{HV}-e(\mathbf{V})\|^2$ corresponds to local area preservation, and $\|\mathbf{CV}-\mathbf{U}\|^2$ represents the position constraints from the handles. Please refer to [20] for details on how to compute each term. Note that $\mathbf{U}$ contains the target positions of the handles. In keyframe editing, $\mathbf{U}$ is changed smoothly because the user moves the mouse continuously. In the remaining frames, $\mathbf{U}$ is computed from handle propagation and may change abruptly.

The above energy can be simplified into the following formula:

$$\min_{\mathbf{V}} \|\mathbf{AV}-\mathbf{b}(\mathbf{V})\|^2 \quad (8)$$

where

$$\mathbf{A}=\begin{pmatrix}\mathbf{L}\\ \mathbf{M}\\ \mathbf{H}\\ \mathbf{C}\end{pmatrix}, \mathbf{b}(\mathbf{V})=\begin{pmatrix}\delta(\mathbf{V})\\ 0\\ e(\mathbf{V})\\ \mathbf{U}\end{pmatrix}.$$

$\mathbf{V}$ consists of two parts: $\mathbf{V}_p$ and $\mathbf{V}_g$. Since $\mathbf{V}_p$ is sampled from the Bézier curves according to Equation (1), it can be represented as a linear combination of the control points of the Bézier curves:

$$\mathbf{V}_p=\mathbf{BP}, \quad (9)$$

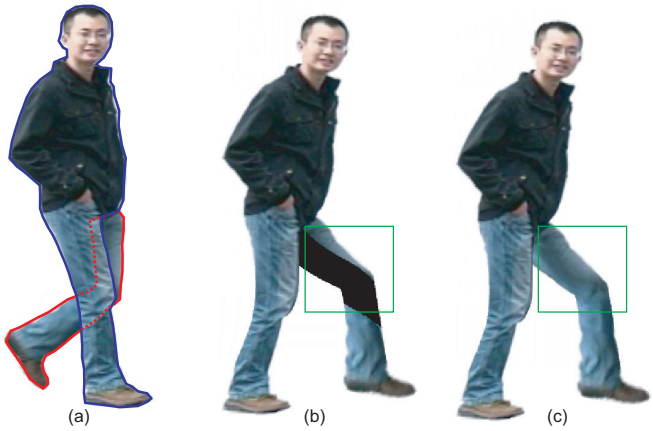

Fig. 5. Handling a video object with self-occlusions. (a) The original shape is modeled as two occluded polygons, one is in red and one in blue. (b) The editing result without video inpainting. The originally occluded region is exposed. (c) The editing result with video inpainting.

where $\mathbf{P}$ is the control point postions and $\mathbf{B}$ is the parameter matrix that maps $\mathbf{P}$ to $\mathbf{V}_p$.

Recall that the interior points $\mathbf{V}_g$ are computed from $\mathbf{V}_p$ using mean value coordinates (Equation (4)), so we have:

$$\mathbf{V}_g=\mathbf{M}_p\mathbf{V}_p=\mathbf{M}_p\mathbf{BP}. \quad (10)$$

Therefore, the point positions $\mathbf{V}$ can be represented as a linear combination of the control points:

$$\mathbf{V}=\begin{pmatrix}\mathbf{V}_p\\ \mathbf{V}_g\end{pmatrix}=\begin{pmatrix}\mathbf{B}\\ \mathbf{M}_p\mathbf{B}\end{pmatrix}\mathbf{P}=\mathbf{WP}. \quad (11)$$

Replacing $\mathbf{V}$ with $\mathbf{WP}$ in Equation (8), we get:

$$\min_{\mathbf{P}} \|\mathbf{AWP}-\mathbf{b}(\mathbf{WP})\|^2. \quad (12)$$

This is a nonlinear least squares problem since $\mathbf{b}$ is a nonlinear function dependent on the unknown control point positions. It can be solved using the inexact iterative Gauss-Newton method as described in [6]. Precisely, the inexact Gauss-newton method converts the nonlinear least squares problem in Equation (12) into a linear least squares problem at each iteration step:

$$\min_{\mathbf{P}^{k+1}} \|\mathbf{AWP}^{k+1}-\mathbf{b}(\mathbf{WP}^k)\|^2, \quad (13)$$

where $\mathbf{P}^k$ is the control point positions solved from the k-th iteration and $\mathbf{P}^{k+1}$ is the control point postions we want to solve at iteration $k+1$. Since $\mathbf{b}(\mathbf{WP}^k)$ is known at the current iteration, Equation (13) can be solved through a linear least squares system:

$$\mathbf{P}^{k+1}=(\mathbf{W}^T\mathbf{A}^T\mathbf{AW})^{-1}\mathbf{W}^T\mathbf{A}^T\mathbf{b}(\mathbf{WP}^k)=\mathbf{Gb}(\mathbf{WP}^k). \quad (14)$$

Note that $\mathbf{G}=(\mathbf{W}^T\mathbf{A}^T\mathbf{AW})^{-1}\mathbf{W}^T\mathbf{A}^T$ only depends on the 2D graph before deformation and is fixed during deformation. It can be precomputed before deformation.

The model reduction we use in Equation (11) is based on the matrices $\mathbf{B}$ and $\mathbf{M}_p$. Both have nice properties: each component of the matrices are positive and the sum of each row

of the matrices equals to one. Therefore, this dimensionality reduction greatly reduces the nonlinearity of $\mathbf{b}$ according to the analysis in [6]. Hence the stability of the inexact Gauss-Newton solver is improved significantly.

Figure 4 shows a comparison between the deformation solver in [20] and our dimension reduced solver. Due to the abrupt change of the handle position, the solver in [20] produces unnatural deformation results, while our solver generates satisfactory results. Please see the companion video for an animation comparison.

3 Handling A Video Object with Self-Occlusion

Video objects in real life often have complex topology. Self-occlusions frequently occur when one part of the object occludes another, especially in articulated video objects. Figure 5 illustrates a simple example. The left leg of the character is occluded by by his right leg.

To enable editing of video objects with complex topology, our system models the shape of the video object with multiple polygons. These polygons may occlude each other. For each polygon, a depth value is assigned by the user to determine its rendering order. With this setting, the user is able to manipulate the meaningful parts of a video object after generating the interior graph for each polygon. However, we still need to solve the following two problems.

First, once the polygons are deformed, some originally occluded regions in the video object may become exposed. However, there is no texture information for these regions in the extracted foreground images. To solve this incomplete texture problem, we adopt an existing video inpainting technique [13] to automatically generate textures for these occluded regions. Figure 5(c) shows the inpainting result. Note the occluded region of the left leg now is filled with inpainted texture.

Secondly, the contour tracking algorithm may output unexpected results when occlusions occur. Although there exist some tracking algorithms that can handle self-occlusions [22], currently we simply decide the positions of the Bézier curves in the occluded regions by interpolating the positions from neighboring keyframes. If the simple interpolation cannot generate satisfactory results, the user can manually adjust the positions of the Bézier curves.

4 Experimental Results

We have implemented the described video object editing scheme on a 3.7Ghz PC with 1GB of memory. For the purpose of clarity, we implement the proposed scheme in two modules: data preparation module and interactive editing module. The data preparation module implements the first two steps, video object cutout and video object shape generation, of the algorithm, and the interactive editing module implements the remaining three steps, keyframe editing, deformation propagation and novel object generation, to create a novel video object (see the algorithm flowchart illustrated in Figure 1). In the following, we will first briefly analyze the time complexity of each module, and then report various experimental results to demonstrate the capability and facility of our scheme.

The output of data preparation is a sequence of 2D graphs and corresponding textures of the video object. The video object cutout is the most time-consuming step in data preparation. Since it needs user intervention and we expect high-quality matting results, it is relatively tedious and usually takes 3-4 minutes for a 100-frame video [11]. The output of data preparation can be stored for arbitrary editing.

For interactive editing, keyframe editing and deformation propagation are two important steps. In keyframe editing, our system runs in real-time due to the high speed of the 2D shape deformation solver [20]. In deformation propagation, we propagate the handle editing results from the keyframes to the remaining frames and then perform offline computation to solve Equation (12) for every frame. The iterative 2D shape deformation algorithm presented in section 2.4 is the most time-consuming step in interactive editing. Here we will analyze its time complexity in detail. The one iteration of deformation algorithm can be formulated into a linear system in Eq. (14). Therefore, its computation also involves two parts: compute function $\mathbf{b}(\mathbf{WP}^k)$ and compute $\mathbf{Gb}(\mathbf{WP}^k)$ to get the new positions of vertices. Precisely, $\mathbf{b}(\mathbf{WP}^k)$ involves linear operations at each graph vertex and $\mathbf{Gb}(\mathbf{WP}^k)$ is just matrix-vector multiplications [20]. Suppose we have M control points and N graph vertices, the time complexity of the deformation algorithm can be easily determined as $O(M^2+N^2)$, which means it is mainly influenced by the number of vertices. The statistics and timings of the interactive editing are listed in Table 1 for the editing results presented in this paper, and the solving time column of Table 1 also proves that the time complexity of our solver is dominated by the number of vertices. The propagation time is just the accumulation of deformation time at each frame.

The convergence curves of the deformation solvers are shown in the Figure 4.c. The reduced solver converges much more faster to an optimal minimum, while the deformation solver in [20] gets stuck in a wrong local minimum. Therefore, the reduced solver significantly improves the speed of deformation algorithm and the quality of the deformation result. Figure 7.b illustrates another comparison between our dimension-reduced solver and the solver in [20]. Note the unnatural deformation result at the top boundary of the teapot from the solver in [20], while the deformation result from our solver is quite natural.

The teapot example in Figure 7 demonstrates the facility of our system. In this example, the user only sets the first frame as a keyfame, deforms it into the desired shape, and specifies the influence area of the keyframe to be the entire sequence. Our system will automatically generate a novel teapot walking sequence (see the accompanying video for the editing process). The handle propagation result after keyframe editing is illustrated in Figure 7.e and Figure 7.f. Since the deformation handle is translated to the new position at a keyframe, the automatically calculated motion trajectory is just the translation of the original motion trajectory.

Two more complicated results with self-occlusions are shown in Figure 6 and Figure 10. In Figure 6, an elephant is

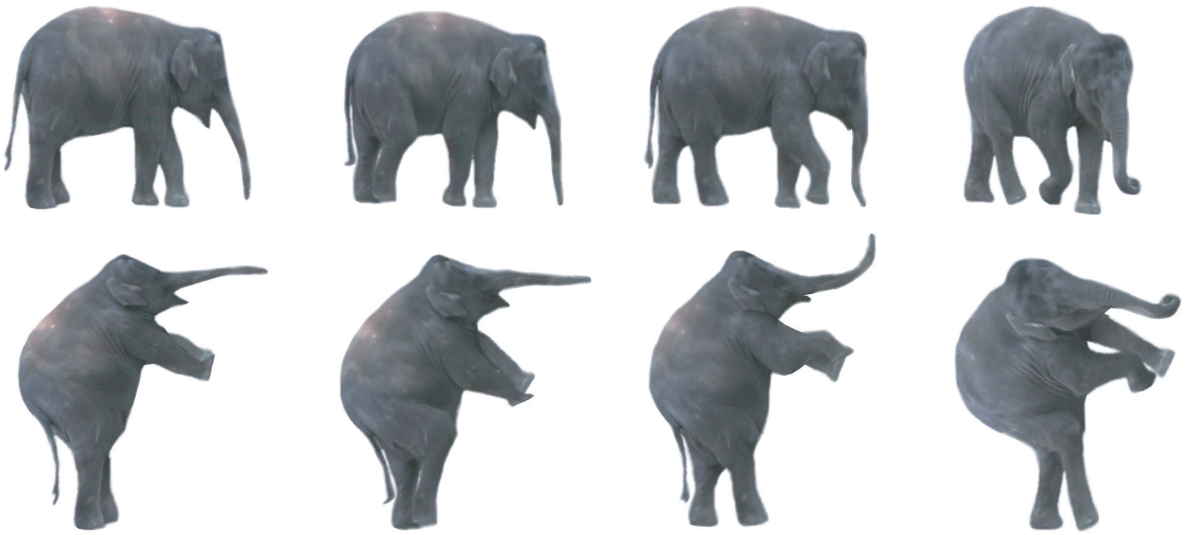

Fig. 6. Deformation of an elephant video object. Top row: original video object. Bottom row: editing result.

Video object	Frames	Bézier curves	Graph vertices	Solving time	Keyframes	Propagation time
Teapot	73	12	698	2.93ms	1	4.69s
Elephant	90	10	1269	4.93ms	7	10.04s
Walking man	61	14	658	2.577ms	6	3.32s
Flame	100	2	519	1.36ms	4	2.998s
Fish	100	5	365	1.44ms	10	3.241s

TABLE 1
Statistics and timings. Solving time means the time for each iteration of our dimension-reduced deformation solver. Keyframes indicates the number of frames edited to achieve the result, and the propagation time means the time required to propagate the keyframe editing to the entire sequence.

made to stand on its hind legs. The large scale deformation in this example is made possible by the power of our dimension reduced deformation solver. Figure 10 demonstrates that our system is capable of editing complex motion, like human walking. The input video object is a man walking on the ground, and we make it walk up stairs.

To achieve the human walking editing result, two consecutive editing steps are performed. The first step is fully automatical and is adapted from the footprint editing algorithm in [21]. Specifically, we select points on the 2D shape to represent the feet of the man (shown in figure 10.e), and then extract footprints from the input video object by checking in what interval the position of points are unchanged or the changes are less than a threshold. Any frame that contains a footprint will automatically become a keyframe. After extracting footprints, the user only needs to draw lines to roughly specify where is the stairs, then our system automatically computes the target position of footprints by projecting them to the specified lines. The handle editing propagation algorithm is then invoked to find out the target position of each foot at each frame. Finally, the video object is deformed at each frame according to the calculated target positions. After this step, the overall motion has been changed to stair walking automatically. Note that the temporal properties of the original motion trajectory are well preserved in the automatically computed motion trajectory with our handle editing propagation algorithm as shown in Figure 10.e and Figure 10.f.

However, the resulting motion might contain visible artifacts, like the compression of the leg, since our deformation solver does not support skeleton constraints. Therefore, we need to perform second editing steps to improve the initial stair walking result. In second step, the user only needs to edit the frames where the artifacts are most obvious, and let the system smoothly propagate the editing to generate the final result. In the walking editing example in Figure 10, only 6 keyframes are edited for a sequence of 62 frames.

The candle flame in Figure 8 exhibits highly nonrigid motions. After editing, all motions of the flame are well preserved. This clearly demonstrates that our system can preserve the temporal coherence of the video object in the original video clip. In Figure 9, both the motion and shape of a swimming fish are changed.

5 Conclusion and Future Work

We have presented a novel deformation system for video objects. The system is designed to minimize the amount of user interaction, while providing flexible and precise user control. It has a keyframe-based user interface. The user only needs to deform the video object into the desired shape at several keyframes. Our algorithm will smoothly propagate the editing result from the keyframes to the remaining frames and automatically generates the new video object. The algorithm can preserve temporal coherence as well as the shape features of the video objects in the original video clips.

Although our deformation system can generate some interesting results, it has several restrictions. First, the handle prop-

agation algorithm requires the 2D shape of the video object to have the same topology, which greatly restricts the motion complexity of the video object. Our method will not work if the shape boundary of the video object experiences topology change in motion. Secondly, the 2D handle editing propagation algorithm does not take the perspective projection effect into consideration, so it may cause undesirable deformation results.

REFERENCES

[1] A. Agarwala, A. Hertzmann, D. H. Salesin, and S. M. Seitz. Keyframe-based tracking for rotoscoping and animation. *ACM Trans. Graphics*, 23(3):584–591, 2004.

[2] M. Alexa, D. Cohen-Or, and D. Levin. As-rigid-as-possible shape interpolation. In *SIGGRAPH 2000 Conference Proceedings*, pages 157–164, 2000.

[3] W. A. Barrett and A. S. Cheney. Object-based image editing. *ACM Transactions on Graphics*, 21:777–784, 2002.

[4] Y.-Y. Chuang, A. Agarwala, B. Curless, D. H. Salesin, and R. Szeliski. Video matting of complex scenes. *ACM Trans. Graphics*, 21(3):243–248, 2002.

[5] M. S. Floater. Mean value coordinates. *Comp. Aided Geom. Design*, 20(1):19–27, 2003.

[6] J. Huang, X. Shi, X. Liu, K. Zhou, L. Wei, S. Teng, H. Bao, B. Guo, and H.-Y. Shum. Subspace gradient domain deformation. *ACM Trans. Graphics*, 25(3):1126–1134, 2006.

[7] T. Igarashi, T. Moscovich, and J. F. Hughes. As-rigid-as-possible shape manipulation. *ACM Trans. Graphics*, 24(3):1134–1141, 2005.

[8] N. Joshi, W. Matusik, and S. Avidan. Natural video matting using camera arrays. *ACM Trans. Graphics*, 25(3):779–786, 2006.

[9] S. Kircher and M. Garland. Editing arbitrarily deforming surface animations. *ACM Trans. Graphics*, 25(3):1098–1107, 2006.

[10] D. H. U. Kochanek and R. H. Bartels. Interpolating splines with local tension, continuity, and bias control. In *SIGGRAPH 1984 Conference Proceedings*, pages 33–41, 1984.

[11] Y. Li, J. Sun, and H.-Y. Shum. Video object cut and paste. *ACM Trans. Graphics*, 24(3):595–600, 2005.

[12] C. Liu, A. Torralba, W. T. Freeman, F. Durand, and E. H. Adelso. Motion magnification. *ACM Transactions on Graphics*, 24(3):321–331, 2005.

[13] K. A. Patwardhan, G. Sapiro, and M. Bertalmio. Video inpainting of occluding and occluded objects. In *Proceedings of IEEE International conference on Image Processing*, pages 69–72, 2005.

[14] S. Schaefer, T. McPhail, and J. Warren. Image deformation using moving least squares. *ACM Trans. Graph.*, 25(3):533–540, 2006.

[15] C. Toklu, A. T. Erdem, and A. M. Tekalp. Two-dimensional mesh-based mosaic representation for manipulation of video objects with occlusion. *IEEE Transaction on Image Process*, 9(9):1617–1630, 2000.

[16] A. Treuille, A. Lewis, and Z. Popovic. Model reduction for real time fluids. *ACM Transaction on Graphics*, 25(9):826–834, 2006.

[17] J. Wang, P. Bhat, R. A. Colburn, M. Agrawala, and M. F. Cohen. Interactive video cutout. *ACM Trans. Graphics*, 24(3):585–594, 2005.

[18] J. Wang, S. Drucker, M. Agrawala, and M. F. Cohen. The cartoon animation filter. *ACM Trans. Graphics*, 25(3):1169–1173, 2006.

[19] J. Wang, Y. Xu, H.-Y. Shum, and M. F. Cohen. Video tooning. *ACM Trans. Graphics*, 23(3):574–583, 2004.

[20] Y. Weng, W. Xu, Y. Wu, K. Zhou, and B. Guo. 2d shape deformation using nonlinear least squares optimization. *The Visual Computer*, 22(9-11):653–660, 2006.

[21] W. Xu, K. Zhou, Y. Yu, Q. Tan, Q. Peng, and B. Guo. Gradient domain editing of deforming mesh sequence. *ACM Trans. Graphics*, 26(3): Article 84, 10 pages, 2007.

[22] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Computing Surveys*, 38(4):1–45, 2006.

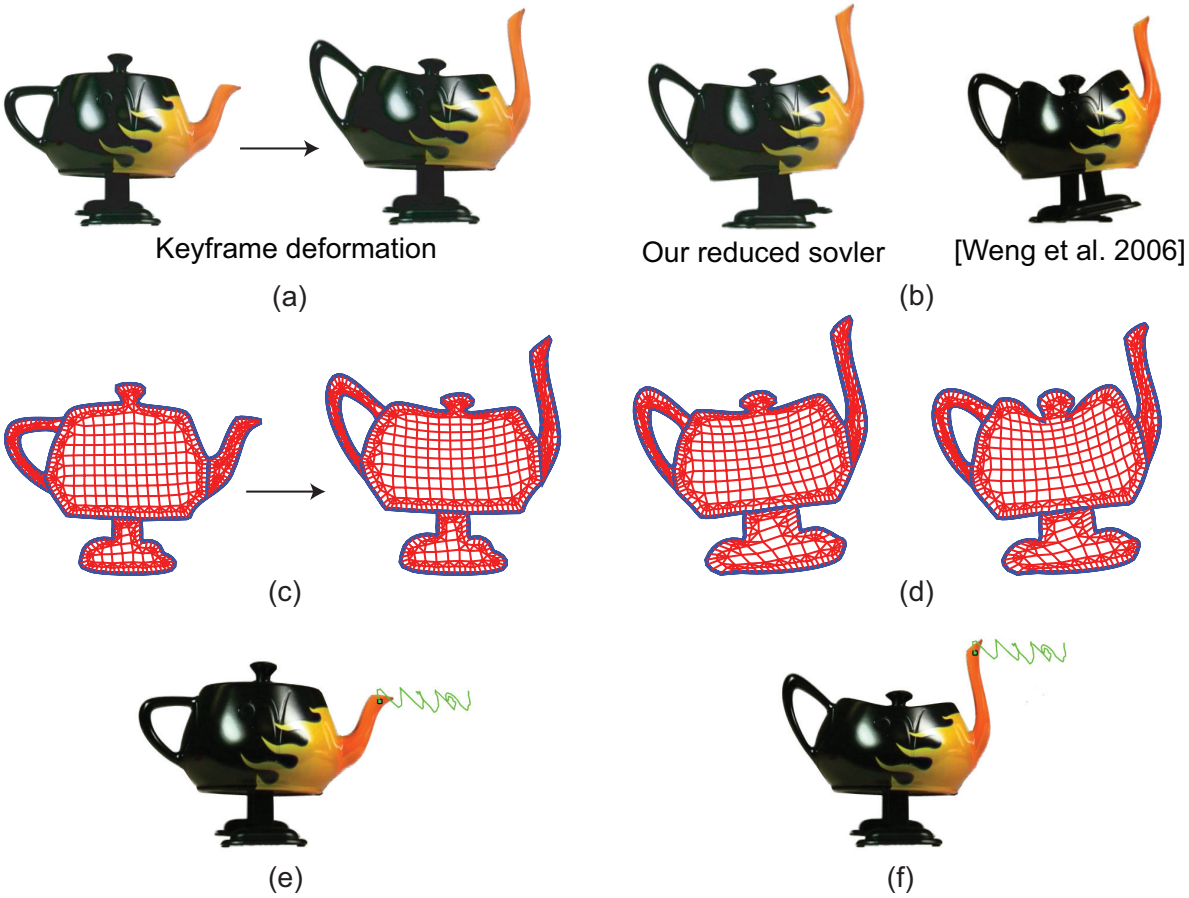

Fig. 7. Deformation of a walking teapot. (a) Keyframe editing. Only one keyframe is edited in this example. (b) Comparison of Propagation result. Note the unnatural deformation result from the solver in [20], while our reduced solver generates natural transitions at the top boundaries of the teapot. (c) 2D graphs corresponding to keyframe editing in (a). (d) Deformed 2D graphs of video object corresponding to the comparison in (b). (e) The motion trajectory of a handle on original video object, indicated by the green curve. (f) The automatically calculated motion trajectory of the handle after keyframe editing.

Fig. 8. Editing of a candle flame. Top row: Original video object. Bottom row: Editing result.

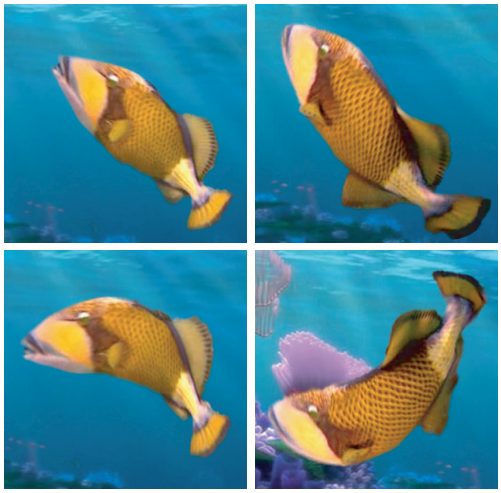

Fig. 9. Editing of a swimming fish. Top row: Original video object. Bottom row: Editing result.

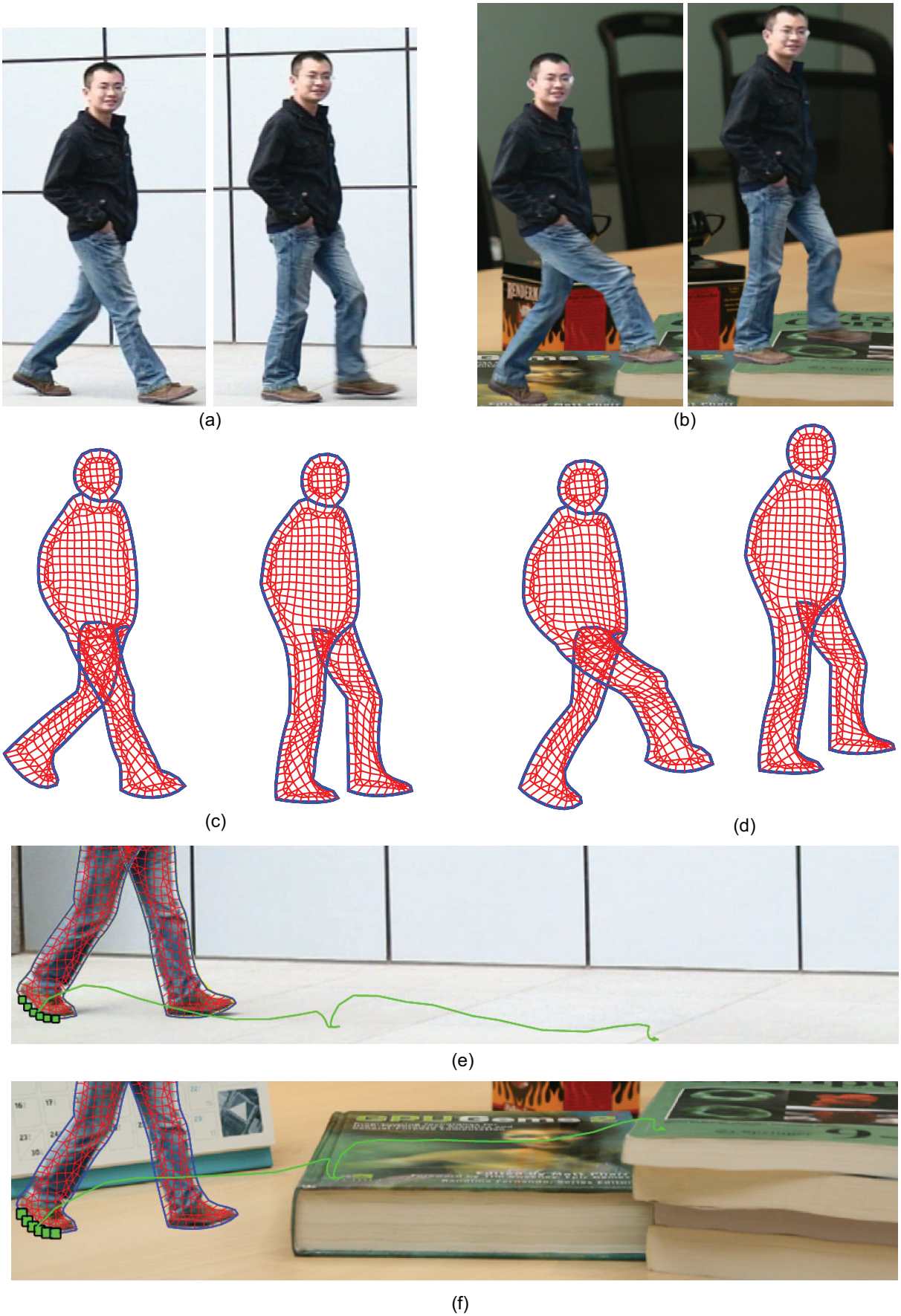

Fig. 10. Walking editing. (a) Two frames in the original video. (b) The editing result. (c) The 2D graphs of the two frames of the original video object. (d) The deformed 2D graphs. (e) The original motion trajectory of the center of the handle representing the right foot, indicated by the green curve. (f) The calculated motion trajectory from handle editing propagation.